ซอฟต์แวร์
ลออ เพิ่มสมบัติ

เริ่มต้นการเขียนโปรแกรม กราฟิก อย่างมีระบบ ตอนที่ 2

ศึกษาวิธีการลากเส้น การวาดรูปเหลี่ยม ในรูปแบบต่าง ๆ ของโปรแกรมกราฟิก

หัวใจของกราฟิกคือการวาดรูป ตามที่เสนอมาแล้วในตอนที่ 1 ในตอนนี้จะเน้นถึงฟังก์ชันในการวาด ซึ่งรวมถึงชุดคำสั่งที่ใช้ประจำในการวาดรูปสี่เหลี่ยม วงกลม วงรี รูปหลายเหลี่ยม และรูปทรงอื่น ๆ เทคนิคการระบายพื้นที่เฉพาะพัฒนาโปรแกรมเพื่อช่วยในการทดลองออกแบบและรูปการระบาย เทคนิคการโปรแกรมโดยใช้ฟังก์ชัน putimage( ) และ getimage( ) ช่วยสร้างภาพเคลื่อนไหว

จุดภาพ (pixels)

ในตอนที่แล้วจุดภาพ เป็นพื้นฐานการสร้างรูปเบื้องต้น เช่น การลากเส้นตรง รูปทรง พื้นผิว หรือวัตถุอื่น ๆ ซึ่ง BGI ได้รวมเอาฟังก์ชันที่เกี่ยวกับจุดภาพไว้สองฟังก์ชันด้วยกัน คือ putpixel( ); /*แสดงจุด 1 จุดบนจอภาพตามตำแหน่งที่กำหนด*/ และ getpixel( ); /*คืนสีให้แก่จุดภาพบนจอภาพ*/ ซึ่งจะลองทำดูทั้งฟังก์ชันที่แสดงจุดภาพ การเข้าถึงจุดภาพ และการแสดงผลออกทางหน้าจอ

การลงจุด 1 จุด

ฟังก์ชัน putpixel( ) จะรับเอาไดออร์ดิเนต x และ y ที่บอกตำแหน่งของจุดภาพบนจอภาพ พร้อมกับรหัสสีฟังก์ชันต้นแบบ คือ

```
void far putpixel(int x, int y, int pixelcolor);
```

ตำแหน่งที่แสดงจุดภาพจะสัมพันธ์กับวิวพอร์ตปัจจุบัน ถึงแม้ว่าช่วงที่แท้จริงของ x และ y ขึ้นอยู่กับขนาดของวิวพอร์ต (ซึ่งตามปกติจะใช้เต็มจอ) ดังนั้นฟังก์ชัน getmaxx( ); และ getmaxy( ); จึงถูกนำมาใช้ในการหาค่า x และ y ที่มีค่าสูงสุด pixelcolor เป็นอาร์กิวเมนต์ที่บอกสีของจุดภาพ คือ เริ่มด้วยสีที่ 0 ถึงค่าที่สูงสุดที่หาได้จากฟังก์ชัน getmaxcolor( );

คำสั่งการวาด

ถึงแม้ว่าคุณจะวาดทุกสิ่งทุกอย่างได้โดยใช้จุดภาพ แต่เทอร์โบซียังมีฟังก์ชันที่อำนวยความสะดวกและง่ายในการใช้ ซึ่งจะแยกเป็นประเภทได้ดังนี้

1. ประเภทเส้นตรง (lines)
2. ประเภทรูปหลายเหลี่ยม (polygon)
3. ส่วนโค้ง วงกลม วงรี (arcs, circles, and ellipses)

การลากเส้น

เส้นตรงจะเกิดจากจุดสองจุดที่มีระยะห่างกันสั้นที่สุด เทอร์โบซีมีให้เราเลือกใช้สามฟังก์ชันเกี่ยวกับเส้นตรงและการเคลื่อนย้ายตำแหน่งของเคอร์เซอร์

ฟังก์ชัน	อธิบาย
line( )	ลากเส้นตรงจากจุด (x_1, y_1) ไปยัง (x_2, y_2)
lineto( )	ลากเส้นตรงจากจุดที่เคอร์เซอร์อยู่ไปยัง (x,y)
linerel( )	ลากเส้นจากตำแหน่งปัจจุบันที่มีทิศทางเกี่ยวเนื่องกับ (dx,dy)
moverel( )	ย้ายตำแหน่งของเคอร์เซอร์ที่อยู่ตำแหน่งปัจจุบันไปปริมาณหนึ่งซึ่งสัมพันธ์กับตำแหน่งปัจจุบัน
moveto( )	ย้ายเคอร์เซอร์ที่อยู่ตำแหน่งปัจจุบันไปยังจุด (x,y)

ตารางที่ 1

ค่าคงที่	ความหมาย	ค่าที่เป็นตัวเลข
SOLID_LINE	เส้นทึบ	0
DOTTED_LINE	เส้นประ	1
CENTER_LINE		2
DASHED_LINE		3
VSERBIT_LINE		4

ตารางที่ 2

ค่าคงที่	ค่าตัวเลข	ความกว้างจุดภาพ
NORM_WIDTH	1	หนา 1 จุดภาพ
THICK_WIDTH	3	หนา 3 จุดภาพ

ตารางที่ 3

ดังแสดงในตารางที่ 1

ลากเส้นโดยใช้โคออร์ดิเนตสมบูรณ์

เราเริ่มด้วยการศึกษาฟังก์ชัน line( ) และ lineto( ) ที่ใช้โคออร์ดิเนตสมบูรณ์ โดยมีโคออร์ดิเนต 2 โคออร์ดิเนตที่กำหนดจุดปลายของเส้นตรง คือ (5,30) กับ (200,180) จะเขียนได้ดังนี้

```
line(5,30,200,180);
```

ซึ่งจะลากเส้นตรงจากจุดโคออร์ดิเนต (5,30) ไปยังจุดโคออร์ดิเนต (200, 180) ดังนั้นถ้าจะเขียนรูปสามเหลี่ยมที่มีจุดมุมยอดอยู่ที่ (x_1,y_1), (x_2,y_2), (x_3,y_3) ก็ต้องใช้คำสั่ง

```
line(x1,y1, x2,y2);
line(x2,y2, x3,y3);
line(x3,y3, x1,y1);
```

การลากเส้นตรงยังมีเทคนิคอื่นอีก คือ การใช้ lineto( ) และ moveto( ) โดยมีโคออร์ดิเนตของจุดเป็นอากิวเมนต์ และไม่จำเป็นต้องส่งค่าคืนกลับ lineto( ) จะใช้ลากเส้นจากตำแหน่งปัจจุบันที่เคอร์เซอร์อยู่ไปยังจุดที่กำหนดใน lineto( ); ตำแหน่งปัจจุบันก็จะถูกปรับให้เท่ากับตำแหน่งที่ระบุไว้ใน lineto( ) moveto( ) ใช้เคลื่อนที่ตำแหน่งปัจจุบัน ส่วนมากจะใช้ในกรณีที่ต้องการเซตค่าเริ่มต้นของส่วนของเส้นตรง ตำแหน่งปัจจุบันเป็นตำแหน่งที่อาจจะถูกกระทบโดยฟังก์ชันอื่น เช่น เมื่อคุณเปิดวิวพอร์ต ตำแหน่งปัจจุบันจะถูกเซตเป็น (0,0) ดังนั้นการวาดรูปสามเหลี่ยมที่เขียนไว้ในตัวอย่างก่อน โดยใช้ moveto( ) และ lineto จะทำได้ดังนี้

```
moveto(x1,y1);
lineto(x2,y2);
lineto(x3,y3);
lineto(x1,y1);
```

ในบางกรณี lineto( ) จะช่วยการวาดรูปที่เป็นลายเส้นสะดวกขึ้น เช่น ถ้าคุณคำนวณจุดสำหรับรูปนั้นขณะตอนที่คุณวาด แต่ลืมทำสำเนาเก็บไว้ก็จะเกิดปัญหา คำสั่ง lineto( ) และ moveto( ) จะช่วยคุณแก้ปัญหาได้มากทีเดียว

การลากเส้นโดยใช้โคออร์ดิเนตสัมพัทธ์

ในการเขียนโปรแกรมประยุกต์ เราจำเป็นต้องลากเส้นโดยใช้โคออร์ดิเนตสัมพัทธ์ เช่น คุณต้องการลากเส้นที่สัมพัทธ์กับจุดหรือเส้น linerel( ) และ moverel( ) จะช่วยคุณได้ ดังนั้นรูปสามเหลี่ยมที่วาดจากตัวอย่างที่แล้วจะเขียนได้ดังนี้

```
moveto(x1,y1);
linerel(x2-x1,y2-y1);
linerel(x_-x2, y3-y2);
linerel(x1-x3,y1-y3);
```

การเซตรูปแบบของเส้นตรง

เราได้กล่าวถึงการลากเส้นตรงวิธีต่าง ๆ กันโดยใช้ BGI BGI อนุญาตให้เราบอกสี ความหนา รูปแบบของ

เส้นตรงตามที่เราต้องการได้ เส้นทุกเส้นจะถูกลากให้เป็นเส้นทึบ ถ้าเราต้องการเปลี่ยนสี ความหนา และรูปแบบ เราอาจจะลากได้โดยใช้รูปแบบที่ผู้ใช้ (ผู้เขียน) กำหนดขึ้นเอง

รูปแบบของเส้นที่นิยมไว้แล้ว

BGI จัดให้มี setlinestyle( ) ที่ใช้เปลี่ยนรูปแบบความกว้างของเส้นที่ลากขึ้น ฟังก์ชันต้นแบบคือ void far setlinestyle (int linestyle, unsigned pattern, int thickness); อาร์กิวเมนต์ linestyle อาจจะกำหนดค่าต่าง ๆ กันได้ 5 ค่า (กำหนดไว้ใน graphics.h) ค่าเหล่านี้ดังแสดงในตารางที่ 2

อาร์กิวเมนต์ตัวที่สองใน setlinestyle( ) ใช้เมื่อผู้ใช้ต้องการสร้างรูปแบบของตนเอง เช่น เลือก 0 เพื่อต้องการเส้นหนา อาร์กิวเมนต์ตัวที่สามทริกเน็ตบอกถึงความหนาที่เส้นตรงทุกเส้นต้องการ ดังแสดงในตารางที่ 3

ถ้าใช้คำสั่ง

```
setlinestyle(DASHED_LINE,0,THICK_WIDTH);
```

จะลากเส้นตรงที่เป็นรอยปะ หนา และใช้สีปัจจุบัน

ตัดสินการใช้รูปแบบเส้นตรงปัจจุบัน

มาดูว่า BGI อนุญาตให้เราทำอะไรบ้างเกี่ยวกับรูปแบบการกำหนดเส้นตรง การเก็บ การดัดแปลง และการเรียกมาใช้ใหม่ BGI มีฟังก์ชัน getlinesettings( ) ช่วยในเรื่องดังกล่าว ส่วนที่ช่วยในการเก็บเส้นตรงคือ linesettingstyle ซึ่งเป็นข้อมูลแบบสตรัค และมีอยู่ใน graphics.h มีหน้าตาดังนี้

```
struct linesettingstyle {
int linestyle;
unsigned upattern;
int thickness;
};
```

โครงสร้างนี้สำคัญเพราะว่า getlinesettings( ) คาดหวังว่าโครงสร้างนี้จะถูกส่งเข้ามา แล้ว getlinesettings( ) จะคืนค่ากลับในรูปของพารามิเตอร์ของแต่ละเส้นตรงนั้น ๆ ซึ่งฟังก์ชันต้นแบบคือ

```
void far getlinesettings
(struct linesettingstype far *lineinfo);
```

ดังนั้นถ้าต้องการดึงเอาเส้นตรงปัจจุบันที่ได้เซตไว้ออกมา ควรใช้ถ้อยแถลง

```
struct linesettingstype savedlineinfo;
/* Declare line structure*/
getlinesettings (&savedlineinfo);
/* Save line settings*/
setlinestyle(savedlineinfo.
linestyle,savedlineinfo.upattern,
savedlineinfo.thickness);
```

การวาดรูปสี่เหลี่ยมผืนผ้า

รูปฟังก์ชันต้นแบบการวาดสี่เหลี่ยมจะอยู่ใน graphics.h โดยมีลักษณะดังนี้

```
void far rectangle (int left, int top, int right,
int bottom);
```

โดยอาร์กิวเมนต์ 4 ตัวจะแทนโคออร์ดิเนตจุดมุมบนซ้ายและจุดมุมล่างขวาของรูปสี่เหลี่ยมผืนผ้า เช่น ต้องการวาดรูปสี่เหลี่ยมผืนผ้า ที่มีจุดมุมบนซ้าย มีโคออร์ดิเนตเป็น (10,50) และมีจุดมุมล่างซ้ายมีโคออร์ดิเนตเป็น (100,150) ก็จะเขียนฟังก์ชันได้ดังนี้

```
rectangle(10,50, 100,150);
```

การวาดรูปหลายเหลี่ยม

ฟังก์ชันที่ใช้ในการวาดรูปหลายเหลี่ยมคือ drawpoly( ) ซึ่งจะใช้อะเรย์ของจุดและการลากส่วนของเส้นตรงระหว่างจุด ด้วยการยึดรูปแบบของเส้นปัจจุบันและลงสีฟังก์ชันต้นแบบคือ

```
void far drawpoly
(int numpoints,int far*polypoints);
```

อาร์กิวเมนต์ตัวแรกต้องการจำนวนโคออร์ดิเนตของจุด อาร์กิวเมนต์ตัวที่สอง ชี้ไปที่อะเรย์ของโคออร์ดิเนต x,y ที่จะเชื่อมกันทำให้เกิดส่วนของเส้นตรง ดังนั้นถ้าต้องการวาดรูปที่มี 3 ด้าน numpoints ควรจะมีค่าเป็น 3 และอะเรย์ของจุดโคออร์ดิเนตจะเป็น 6 ซึ่งสอดคล้องกับจุดปลายทั้งสองของเส้นตรง

สมมติว่าจุด 3 จุดคือ (10',30), (300,30) และ (100,90) จะวาดรูปได้โดยการกำหนดอะเรย์ของจุดได้ดังนี้

```
int points[ ]={10,30,300,30,100,90};
drawpoly(3,points);
```

หมายเหตุ

เราใช้ชื่อของอะเรย์ในการส่งผ่านอะเรย์ไปยังฟังก์ชันได้ แต่การใช้ฟังก์ชัน drawpoly( ) ยังไม่ทำให้รูปหลายเหลี่ยมเป็นรูปหลายเหลี่ยมที่ปิด ถ้าต้องการให้เป็น

รูปหลายเหลี่ยมปิด ควรเพิ่มโคออร์ดิเนตไปอีก 1 จุด คือเพิ่มจุดแรกในที่นี้คือ (10,30) และเพิ่มจำนวนจุดอีก 1 เป็น 4 ดังนี้

```
int points[ ]={10,30,300,30,100,90,10,30};
        drawpoly(4,points);
```

ส่วนโค้ง วงกลม และวงรี

ฟังก์ชัน arc( ); ใช้วาดเส้นโค้ง

ฟังก์ชัน circle( ); ใช้วาดวงกลม

ฟังก์ชัน ellipse( ); ใช้วาดวงรี

ฟังก์ชัน pieslice( ); วาดรูปขนมพาย (วาดรูปส่วนของพายพร้อมระบายสีและแบบ)

ฟังก์ชัน fillellipse( ); ระบายสีรูปวงรี

ฟังก์ชัน sector( ); วาดรูปสามเหลี่ยมฐานโค้ง

ฟังก์ชันต้นแบบของ arc( ) คือ

```
void far arc(int x, int y, int stangle,
int endangle, int radius);
```

สองอาร์กิวเมนต์แรกบอกถึงจุดศูนย์กลางของส่วนโค้ง stangle และ endangle เป็นมุมเริ่มต้นและมุมสุดท้ายของส่วนโค้ง ซึ่งวัดเป็นองศาที่ทวนเข็มนาฬิกา โดยเริ่มที่เข็มนาฬิกามีเข็มยาวชี้ที่เลข 12 และเข็มสั้นชี้ที่เลข 3 ส่วนอาร์กิวเมนต์ตัวสุดท้ายคือรัศมีของวงกลม โดยวัดจากจุดศูนย์กลางของส่วนโค้งมีความยาวเท่ากับจำนวนจุดภาพ

การหาจุดปลายของส่วนโค้ง

บางครั้งถ้ารู้ว่าปลายของส่วนโค้งสิ้นสุดตรงไหนจะทำให้สะดวกในการวาด เช่น ถ้าเราต้องการจะเชื่อมจุดปลายของโค้งเข้าด้วยกันโดยอาศัยเส้นตรง ควรใช้ฟังก์ชัน getarccoords( ) ที่

```
/*hemi.c--Draw a hemisphere at the center of the screen.
*/
#include <graphics.h>
#include <conio.h>
main()
{
 int gmode;
 int gdriver=DETECT;
 struct arccoordstype arccoords;

 initgraph(&gdriver,&gmode,"");
 arc(getmaxx()/2,getmaxy()/2,0,180,getmaxx()/4);
 getarccoords(&arccoords);
 line(arccoords.xstart,arccoords.ystart,
            arccoords.xend,arccoords.yend);
 setcolor(WHITE);
 circle(200,200,50);
 floodfill(200,200,WHITE);
/* setcolor(RED);
 circle(250,250,50);
 floodfill(250,250,RED);*/
/*  clrscr();
 setcolor(YELLOW);
 circle(250,250,50);
 floodfill(250,250,YELLOW);*/
 getch();
 closegraph();
 return(0);
}
```

รูปที่ 1

BGI มีอยู่ ฟังก์ชันต้นแบบของ getarccoords( ); คือ

```
void far getarccoords
(struct arccoordstype far *arccoords);
```

ซึ่งจะเห็นว่ามี 1 อาร์กิวเมนต์เท่านั้นที่เป็นโครงสร้างชนิด arccoordstype ที่เก็บจุดปลายของส่วนโค้งสุดท้ายที่วาดขึ้นกับตำแหน่งของจุดศูนย์กลางที่นิยามไว้ใน graphics.h มีรูปแบบดังนี้

```
struct arccoordstype {
    int x,y;
  int xstart,ystart;
    int xend,yend;
};
```

เช่น สมมติว่าต้องการวาดรูปครึ่งวงกลม เราอาจใช้ arc( ) และ line( ) ผสมกัน การเชื่อมส่วนโค้งด้วยส่วนของเส้นตรงเราจะใช้ getarccoords( ) เพื่อหาว่าจุดปลายของโค้งอยู่ที่ใดจะได้

ลากเส้นเชื่อมได้ถูก ดังโปรแกรม hemi.c ในรูปที่ 1

พื้นฐานของการสร้างภาพเคลื่อนไหว

getimage( ) และ putimage( ) ใช้ในการสร้างเนื้อที่ที่เป็นรูปสี่เหลี่ยมผืนผ้าบนจอภาพ สามารถจะตัด ปะ เคลื่อนย้าย หรือเปลี่ยนเนื้อที่บนจอภาพโดยไม่ต้องกังวลกับเนื้อที่ในหน่วยความจำแต่อย่างใด ฟังก์ชันต้นแบบของ getimage( ) คือ

```
void far getimage(int left,int right,
int bottom, void far *bitmap);
```

อาร์กิวเมนต์ 4 ตัวแรกบอกถึงขอบเขตบนซ้าย ล่างขวาของจุดภาพที่จะถูกทำสำเนาไว้ อาร์กิวเมนต์ตัวที่ 5 จะชี้ไปยังจุดที่จัดเก็บไว้ในรูปอะเรย์ (บิตแม็พ) ขนาดของอะเรย์บิตแม็พขึ้นอยู่กับขนาดของ สกรีนอิมเมจที่ได้ทำสำเนาไว้และโหมดกราฟิกปัจจุบัน

จงจำไว้ว่าแต่ละโหมดสนับสนุนจอภาพที่มีความละเอียดต่าง ๆ กัน และต้องการขนาดความจำที่ต่างกัน การหาขนาดของภาพบนจอ BGI มีฟังก์ชัน imagesize( ) เช่น เราต้องการทำสำเนารูปที่มีขอบเขต (10,10) และ (100,100) จำเป็นต้องประกาศและเข้าถึงเนื้อที่ของบิตแม็พ ดังนี้

```
void *screenimage;
screenimage = malloc(imagesize
(10,10,100,100));
```

อิมเมจจะถูกทำสำเนาไว้ในสกรีน-อิมเมจด้วยการเรียก

```
getimage( );
getimage(10,10,100,100,screenimage);
```

จะได้สำเนาเก็บไว้ในอะรย์บิตแม็พ และสามารถจะทำสำเนาไว้ต่างที่กันบนจอภาพด้วยฟังก์ชัน putimage( )

ฟังก์ชันต้นแบบของ putimage'( ); คือ

```
void far putimage(int left,int top,
void far *bitmap,int op);
```

อาร์กิวเมนต์ 2 ตัวแรกบอกตำแหน่งบนซ้ายซึ่งอิมเมจจะถูกผ่านไปยังฟังก์ชัน และจะเห็นว่าฟังก์ชันนี้ไม่ต้องการขอบเขตของมุมขวาล่าง อาร์-กิวเมนต์ตัวสุดท้ายของ putimage( ) คือ op บอกถึงวิธีการที่อะเรย์บิตแม็พทำสำเนาออกจอภาพ (ดังตารางที่ 4)

เช่น ถ้าต้องการทำสำเนาบิตแม็พ-อิมเมจไปยังตำแหน่ง (110,10) บนจอภาพ ก็ใช้คำสั่ง putimage (110,10, screenimage,COPY_PUT); ในทำนองเดียวกัน ถ้าต้องการอินเวอร์สอิมเมจของส่วนที่ทำสำเนาไว้แล้ว ก็ใช้คำสั่ง

```
putimage(10,10,screenimage,NOT_PUT);
```

ค่าคงที่	ค่า	อธิบาย
COPY_PUT	0	ทำสำเนา bit image ออกจอภาพ
XOR_PUT	1	Exclusive-or bit image and screen
OR_PUT	2	Inclusive-or bit image and screen
AND_PUT	3	And bit image and screen
NOT_PUT	4	copy inverse of bitmap to screen

ตารางที่ 4

```
/* PROGARM #1 */
#include <graphics.h>
main( )
{
int graphdriver = DETECT,graphmode;
void *buffer;
unsigned size;
intgraph (&graphdriver, &graphmode, " ");
size = imagesize(0,0,20,110);
buffer = malloc(size);
getimage(0,0,20,10,buffer);
putimage(0,0,buffer,COPY_PUT);/*restore bit*/
free(buffer);
closegraph( );
}
```

รูปที่ 2